DEVELOPMENT OF THE

SYMBOLIC PROCESSING ALGORITHMIC RESEARCH COMPUTER

- SPARC -

SEMI-ANNUAL REPORT

REPORTING PERIOD: 78 JUNE 01 - 78 NOVEMBER 30

Peter G. Juetten, Project Engineer

Phone: 612/853-3713

Contract Effective Date: 79 June 01

Contract Expiration Date: 79 April 30

This research was sponsored by the
Defense Advanced Research Projects
Agency under ARPA Order No.: 3601
Contract No.: MDA903-78-C-0288
Monitored by: Dr. John Neff, AFOSR/NE

Control Data Corporation
DIGITAL IMAGE SYSTEMS DIVISION
2800 East Old Shakopee Road
Minneapolis, Minnesota 55420

SUMMARY

This report summarizes progress on development of the SPARC processor which was started by ARPA in 1977. This is a joint research effort in computer architecture by Carnegie Mellon University and Control Data Corporation to develop a processor concept which anticipates future image processing needs. This processor ~~is being~~ was designed, using state-of-the-art technology and automated design techniques, as an add-on for general purpose host computers. The tasks are divided such that CMU is responsible for developing user system software aids and Control Data is responsible for the hardware development and basic operating system, along with the development of more extensive system software. This phase of the development program is expected to be completed in early fall of 1979, by which time both CMU and CDC will have SPARC processors installed in their respective laboratories. At the end of November, 1978, the processor design was approximately 75% complete. About half the electronic parts have been placed on order, along with cabinetry for both the CMU and CDC processors. In addition, work ~~is~~ was also in progress at CDC to develop a multiple processor architecture which utilizes extended versions of the SPARC processor. A key feature of this new architecture is a flexible, high-bandwidth, inter-processor communication network known as the ring system. Initial requirements definitions and preliminary reference manuals for a microcode cross assembler and a register-level simulator have been written.

INTRODUCTION

It is anticipated that current capabilities to process images for military applications will soon be inadequate. These increasing demands for image processing power can be attributed to three major factors. First, more imagery data is being collected to produce a greater number of processed images. Second, an increasing fraction of collected imagery data is being sensed or recorded in digital form, and finally, greater intelligence on the part of the image processing system is being demanded. These changes will require improvements of image processing systems in both signal-level processing and symbolic-level processing.

Signal-level processing is largely a matter of arithmetic manipulations, comprised of tasks such as transformation, interpolation, and filtering. These signal-level tasks are readily supported by currently available numeric signal processors. To some degree, increasing signal-level demands can be met by brute force increases in computer power with networks of available processors. Unfortunately, increases in the number of digital imagery sensors and image users tend to suggest that processing demands should increase by two orders of magnitude over the next few years. Meeting such an increase will have to be accomplished by both an increase in signal-level performance and a shift of some of the burden from the signal-level to the symbolic-level.

Symbolic-level processing is concerned with the detection of relationships among entities having certain semantic attributes, and involves symbol manipulating tasks such as searching, decision making, and path finding. Available numerically-oriented processors are poorly suited for these tasks. This is particularily true of high-performance signal processors which rely heavily on pipelining, since their pipelines generally have to be flushed and refilled with each decision branch. While many non-numeric processors have been proposed and a few built, there are two important disadvantages to attempting to satisfy increasing image processing demands through a network of numeric and non-numeric processors. First, the net would be inhomogeneous, leading to problems in hardware maintenance, and probably severe problems in interfacing and software development and change. The second disadvantage is major and relates to the difficulty of smoothly shifting the processing burden from the signal level to the symbolic level.

The preceeding observations suggest a need for a processor with both improved signal level computing power and high performance at the symbolic level of image processing. This report deals with the current status of an architecture research project being performed jointly by Control Data Corporation and Carnegie Mellon University to develop such a processor. At this point in time, the processor detailed design is approximately 75% complete, and work is beginning on various elements of software to be used with the processor.

TECHNOLOGY

There are two basic approaches towards improving the performance and cost-effectiveness of a digital processor. Gains in processing speed may be achieved through the use of new component technology, providing an increase in the basic speed of the processor building block or logic gate and/or a decrease in signal interconnect time through the use of large scale integration techniques. Such an approach is relatively straightforward, assuming the technology is available. A second method of securing improved processor performance is through the use of design and architectural improvements. Such improvements must be firmly based on experience gained in past development programs and systems applications.

The SPARC processor design utilizes both methods to attack the problem of increased performance. Early in the program a technology investigation was conducted; the results of which indicated the most cost-effective technology for next generation image sensor processing equipment to be an ECL LSI technology which has been developed by Control Data Corporation over the past five years in conjunction with device fabrication performed by Motorola and Fairchild.

Some significant features of this technology are presented in Table 1. The fabrication process provides a semi-custom version of LSI. With this process, the cost of fabricating new chip types is much lower than with a completely custom LSI. In this technology, the diffusion pattern is fixed, and is the same for each chip type. Only the top two layers of metalization interconnect are used to provide the variable structure. The basic internal propagation delay utilizing this technology is approximately 750 pico seconds per gate, a factor of 4-5 improvement over commonly used TTL technology. The level of integration, 168 2-input gates per array, while being considerably smaller than that available with MOS type technologies, is significantly higher than that which is commonly available in high performance technology such as ECL. Thus, logic packing density is increased with the corresponding decrease in gate interconnect distance and delay, resulting in performance gains.

TABLE 1. ECL LSI ARRAY DESCRIPTION

- 168 ECL GATES PER ARRAY
- 2 INPUT AND-NAND GATES
- 165 x 175 MIL DIE
- 4 GATES PER CELL
- 48 SIGNAL PINS
- EXTERNAL GATES
- 8 LOADS PER OUTPUT
- COLLECTOR DOTTING
- EMITTER "AND"
- SUBNANOSECOND GATE DELAY
- 1-5 WATTS

A new printed circuit board design was also developed by Control Data for use with the ECL LSI arrays. The basic element is a 10 x 12 inch, 15-layer circuit board which may be used to hold up to 150 LSI arrays. A significant feature of this board is the inclusion of the signal line terminating resistors required by the ECL technology as a buried layer in the LSI board itself. Thus, no valuable board real estate need be used for these resistors, as is common in other ECL designs where the line terminations are housed in dual or single in-line packages or even as discrete components. The use of the board termination resistors has been estimated to increase available board real estate by as much as 50%, while providing a relatively quiet, controlled impedance environment for signal transmission.

Due to a rather high power dissipation of the ECL LSI arrays (1-5 watts per array) an efficient heat transfer mechanism is required to remove the power generated by the electronics. This is achieved in the ECL LSI technology through the use of a freon refrigeration system. Each LSI array, when mounted on the circuit board, rests directly upon a copper tube through which freon refrigerant is continually pumped. The cooling system, therefore, is an integral part of the circuit board assembly. This refrigeration system allows relatively uniform device operating temperatures to be maintained throughout the system. Junction temperatures on the order of 45-50 degrees Centigrade are achieved, thus increasing component reliability significantly.

Most system designs based on the LSI technology also make use of lower power compatible technology, such as SSI and MSI parts of the Fairchild F100K logic family, and various high-performance ECL RAMs. These devices, in general, dissipate less power than the LSI arrays, typically on the order of half a watt per chip. They therefore do not require the intimate contact with the refrigeration system that the LSI arrays do. In order to efficiently package these devices, an auxiliary circuit board was designed which effectively allows up to 160 of these lower power devices to be mounted in the same area that 15 LSI arrays would occupy. The SPARC processor makes use of both ECL LSI arrays and of SSI, MSI, and RAM devices mounted on auxiliary boards.

Another key element of this technology is that the machine is completely simulated at the gate level prior to fabrication, utilizing extensive automated design and simulation software packages developed by CDC. The effectiveness of this approach has been drastically demonstrated in a test bed project in which checkout time following assembly was reduced to a few weeks. This contrasts greatly with the normal checkout of a new computer which can take several months, if not years. Some features of the simulation and other automated design tools used in the SPARC project are listed in Table 2.

All of this technology has been previously developed by Control Data Corporation to provide a basis for future lines of high-performance, general-purpose computers. When the SPARC project was begun, the technology had already been demonstrated, and was being used as the basis of several advanced versions of CDC commercial machines. It therefore was available to be used as the basis for the SPARC processors.

PROCESSOR ARCHITECTURE

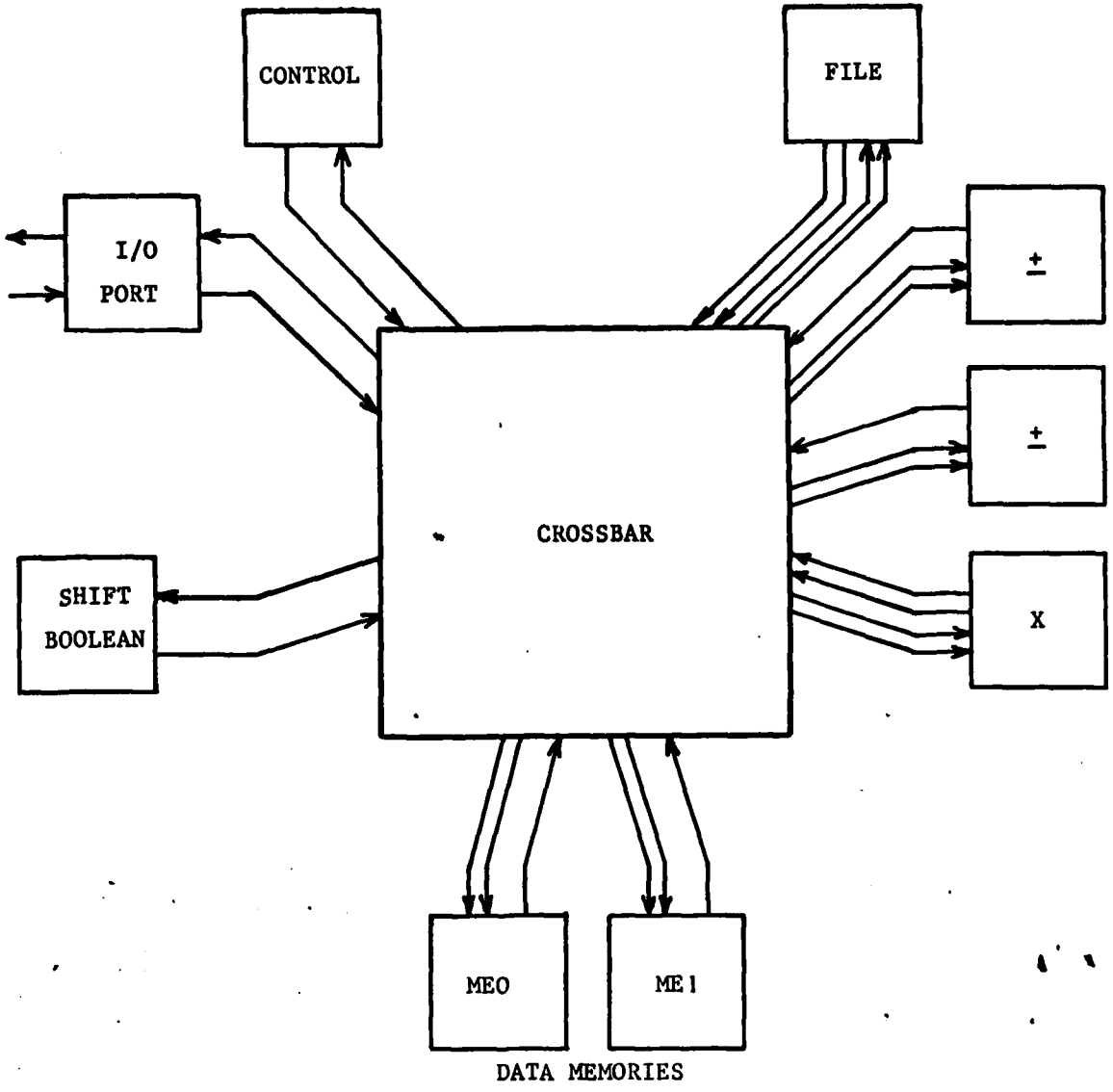
The basic processor block diagram, as shown in Figure 1, consists of a collection of relatively autonomous functional units which communicate with each other via a generalized interconnection mechanism which can be thought of as a crosspoint switch.

Processors such as SPARC achieve their performance rate through the use of a high degree of internal parallelism. That is, they possess the ability to perform several operations at a time. Theoretically, each processor functional unit may be performing a different operation at the same time. Since it is relatively unusual for the result of an operation performed in a particular functional unit to be utilized repetitively by the same unit, a means must be provided to transfer the results from one functional unit to another.

TABLE 2. AUTOMATED DESIGN TECHNIQUES

- LOGIC SIMULATION
  - 75,000-100,000 GATES
  - 24 ARRAY TYPES
  - 10 BOARDS
  - GATE, FOIL, COAX DELAYS TO 10 PS
  - WORST CASE VARIATION
  - ASSIST
- BOARD ROUTER
  - CYCLICAL
  - INTERACTIVE
  - PHOTOPLOT TAPE
  - SIMULATOR INPUT
- ARRAY GENERATOR
  - DIGITIZED ARRAY LAYOUT
  - SPACING CHECKS
  - MASK GENERATION TAPE
  - SIMULATOR INPUT

Figure 1. Basic SPARC Architecture



Several approaches have been historically used to provide this internal data transfer capability. One of the more common is to provide an internal data bus, or combination of busses, joining together many of the functional unit inputs and outputs. However, as the number of internal functional units in a processor increases, such data busing schemes begin to suffer bandwidth limitations, since only one transmitter on a given bus may be active during a given time period. Thus, the effective amount of parallelism realizable in practice in the machine tends to be limited by the bandwidth available for transmitting results. Another classic mechanism for providing inter-unit transfer capability is through the use of a central register or register file, from which all functional units obtain input operands, and into which all results are returned. This architecture, while attractive from several standpoints, as the amount of parallelism in the machine is increased, also begins to suffer from bandwidth problems. If the register file is implemented using small, extremely fast random access memory devices, it is constrained to performing only one operation at a given point in time. That is, only one unit may be obtaining operands or delivering results at a time. Implementations using individual registers or flip-flops, providing multiple, simultaneous read/write capability may be devised. However, these tend to become exceedingly cumbersome and bulky as attempts are made to provide more and more parallelism.

A third alternative is to interconnect the processor's functional unit by means of a generalized crossbar switching network, through which large quantities of data may be simultaneously routed from multiple sources to multiple destinations. Such a network offers the advantage of extremely high data transfer bandwidths, as well as complete generality of functional unit interconnection. While such generalized interconnection schemes become prohibitedly expensive in terms of hardware reuqired when the total number of interconnections becomes large, this penalty is not as severe when a limited number of data sources and sinks are considered. The SPARC processor implements a 16-input, 18-output crossbar network which contains about 20% of the total amount of the processor hardware. This network is capable of routing any of sixteen-bit input quantities to any of the 18 destinations on a given clock cycle. Thus, the theoretical internal data transfer bandwidth of the SPARC

processor approaches 12.8×10^9 bits per second, significantly higher than that which may be achieved through data bussing or register file approaches. The implementation of such a network lends itself nicely to the LSI technology described above, and the entire network provides a data transfer bandwidth capable of supporting the parallelism of the processor at a reasonable cost in terms of hardware complexity.

A block diagram of a typical SPARC functional unit is shown in Figure 2. The units perform such basic functions as integer addition/subtraction, shift and masking operations, bit-by-bit logical operations and multiplication. Additional units provide temporary storage mechanisms for operands and results, while still other functional units implement processor input/output me· .isms. Each unit contains input registers which are controlled from the micr(struction word. All the units, with the exception of the multiply hardware in RC, perform their designated operations and deliver results via the cross .r switch to the input register in another functional unit in one machine cycle time. In addition, most of the units contain hardware to compare the result of the operation to a previously defined quantity held in an auxiliary register in the unit. The results of these comparisons are available by the processor control mechanism for use in branching and decision making.

A processor containing a relatively large number of functional units, such as SPARC, which are capable of simultaneous operation and are interconnected by a generalized high-bandwidth data transfer path, must be supported by a control structure capable of preserving order amongst the internal operations and data paths. Furthermore, in a processor such as SPARC in which the individual operations performed by the functional units are for the most part rather simple, and capable of being executed in the same amount of time as required to fetch control instructions from the memory, a large control bandwidth is required. The control mechanism must be capable of supplying information to a large number of functional unit and data routing mechanisms from each instruction if maximum use is to be made of the inherent parallelism available. In addition, it is desirable to have the format of the control mechanism to be quite general and flexible, capable of

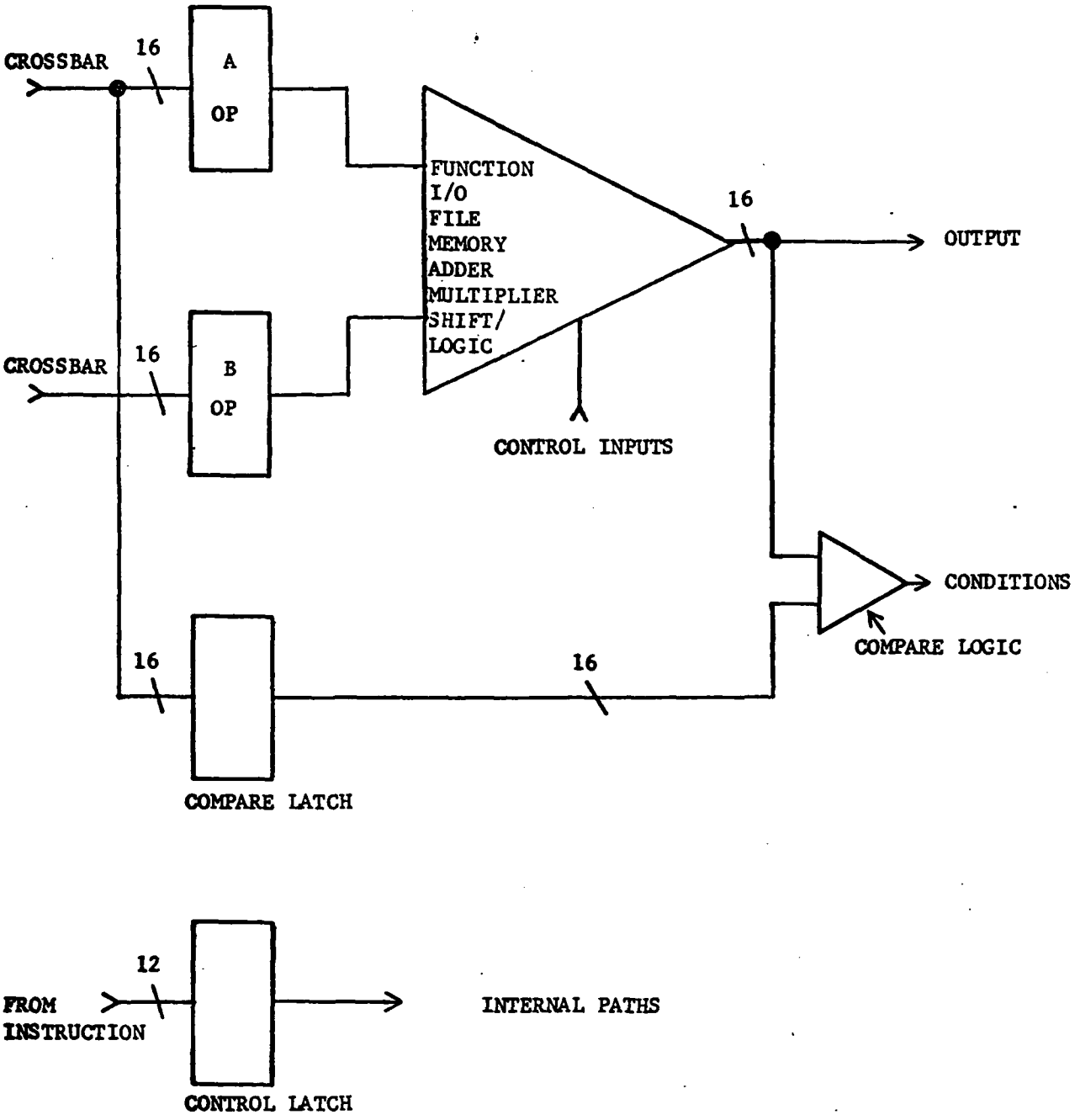
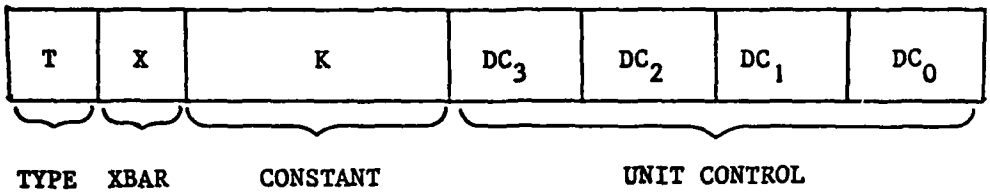


Figure 2. Functional Unit

accommodating and controlling a variety of functional units, some of which have not yet been conceived. To accomplish these goals, the instruction format shown in Figure 3, as the original SPARC instruction format was devised. Four identical fields in the instruction were to be devoted to functional unit control. Information intended for any functional unit could be presented in any of the four control fields, identified by the portion of the field containing a unit tag. Information in these dynamic control fields would include functional unit tags, information specifying the operation which was desired to have the unit perform, and bits to control the clocking of data into selected functional unit input registers. A K or constant field was provided to give the ability to insert constants from the microinstruction into the internal processor data stream, as well as providing base addresses for branching instructions. Control of the data routing mechanism in the crossbar was to be provided via a small, high-performance, second-level control memory which was indirectly addressed by the X field in the SPARC microinstruction. This memory, 72-bits wide, provided all the information necessary to specify crossbar routing for a given instruction cycle. This memory was intended to be sixteen words deep, thus allowing any of sixteen crossbar patterns to be specified by a 4-bit in the microinstruction. Other instruction types were to be utilized for loading and unloading the instruction memory and the crossbar memory. This control structure was documented early in the SPARC program and released to software analysts for investigation and trial coding of selected algorithms.

As this analysis progressed, certain deficiencies in the proposed instruction format were identified. The first problem became evident in the area of functional unit control. The instruction format described allowed control of only four functional units from each microinstruction, out of a possible total of sixteen, which might be included in the processor. Data could only be moved to a functional unit through use of one of the control fields. Since most of the SPARC units complete their operations in a single instruction cycle time, this control structure forced a large percentage of the functional units to be idle at any given moment, and resulting in relatively inefficient use of processor hardware.

ORIGINAL SPARC INSTRUCTION FORMAT



FINAL SPARC INSTRUCTION FORMAT

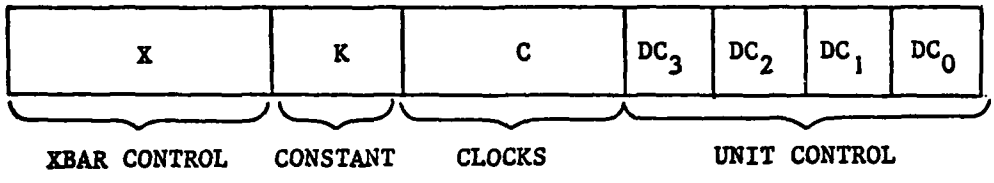


Figure 3. SPARC Instruction Format

An obvious solution would be to increase the processor control bandwidth by including more dynamic control fields in an instruction, up to a theoretical maximum of sixteen. Since each functional unit was felt to require approximately twelve bits of control information, this solution would lead to a prohibitively wide microinstruction word. Therefore, another approach was developed. The software analysis work showed that in the normal pipeline method of coding algorithm kernels, while it was desirable to be able to pass data through functional units and have them perform an operation at a clock cycle rate, in many cases it was not necessary to actually change the operation which the unit was performing at the same rate. That is, a certain number of program initialization steps could be performed in which adders could be instructed to add or subtract, memories instructed to index in a certain manner, shift counts established, and the like, after which data could be streamed from unit to unit simply by controlling the clocks of the unit input registers and the crossbar routing mechanism. To this end, the functional unit input register clock control bits were moved outside of the dynamic control field and placed in a section of the microinstruction dedicated to this function. Utilizing the final SPARC instruction format as shown in Figure 3, incorporating this modification, it is now possible to change the operation taking place in four functional units in a single cycle, and furthermore, to control data routing to all of the functional units in the same microinstruction. This has resulted in greatly increased processor performance and much more efficient use of the hardware, with the number of functional units in simultaneous use limited only by the ingenuity of the programmer.

A second problem with the instruction format was found to be related to the depth of the crossbar control memory. While the sixteen words or patterns available generally proved to be sufficient to handle an algorithm kernel, problems arose as the kernels were nested together into actual programs to provide useful functions. The problem was further complicated by the use of generally coded subroutines to perform functions for multiple uses, and grew increasingly worse when interrupt programming was considered. It quickly became apparent that a sixteen-word crossbar memory was inadequate. One solution to this problem was to increase the depth of the crossbar control memory. There proved to be several technical problems with this approach,

basically related to the size of memory chips available which would meet the performance required of such a second level memory. After examining several possible approaches, the final solution to this problem for the SPARC processor evolved as the complete elimination of the second-level crossbar control memory, appending the XBAR control field to the microinstruction instead. This solution, of course, increases the total number of memory chips required in a processor. However, it simplifies the hardware design by eliminating the second-level memory and its associated addressing, timing, and control mechanisms. This solution completely eliminates all programming problems associated with the limited depth of crossbar control, since each and every microinstruction has a complete crossbar control field associated with it.

Thus, the final SPARC instruction format evolved. Control of functional unit operation is contained in four completely general dynamic control fields. The movement of data through the units is controlled by dedicated clock bits in a separate field in the microinstruction. The constant/branch address field remained. The type and crossbar memory address fields of the original microinstruction have been eliminated and replaced by a wide field, giving complete control of the data routing mechanism during each instruction execution time.

The capabilities of the various SPARC functional units are given in Table 3. The adder and multiplier units can perform operations on 16-bit operands, or can treat their inputs as two sets of 8-bit byte operands. In the case of the multiplier, this means that two independent 16-bit results can be produced simultaneously by two 8-bit multiplier operations. A 32-bit result is produced when the operands are treated as 16-bit words. The adder and shift/boolean units are capable of handling operands longer than sixteen bits. In the case of SPARC, the adder units can perform 32-bit additions or subtractions in each instruction cycle. Data memories are treated by the control structure of the processor in a manner identical to other functional units. Each memory has a capacity of 1,040 16-bit words, and performs a read or write operation in one machine cycle time. The memories are provided with several different modes of addressing, including automatic indexing. A general purpose register file unit has been included to provide temporary storage for operands. In signal level image processing tasks, where several functional units may be connected to form

TABLE 3. FUNCTIONAL UNIT CHARACTERISTICS

UNIT	NO.	OPERATIONS	DATA TYPES *
ADDER	2	1's Complement 2's Complement Increment Double Merge Bytes	Word Dual Byte Double Word **
MULTIPLIER	1	2's Complement Magnitude Cross-Byte	Word Dual Byte
SHIFT/BOOLEAN	1	Right Shift Right Circulate 0-15 Positions 16 Boolean FNS.	Word Double Word **
FILE	1	Simultaneous Read & Write 8 Word Capacity	Dual Word Double Word
Data Memories	2	Read or Write 1024 Word Capacity 16 Indices Direct Indirect Post-Increment Post-Decrement Post-Add Constant Post-Subtract Constant Address Compare	Word Double Word **
RING PORT	1	Ring I/O 16-Word Input Buffer 16-Word Output Buffer Connects to all Files, and switches	Word

* 16 BITS PER WORD, 8 BITS PER BYTE

II WITH TWO FUNCTIONAL UNITS OF THIS TYPE

pipelines, the file unit has been found to be extremely useful in realizing the small delays necessary in programming tight pipes. Basic processor I/O is handled through the functional unit called the Ring Port. This port allows multiple copies of the processor to be connected together by means of a ring communcications network. It also serves as a vehicle to allow the SPARC processor to interface with PDP-11 or other types of external equipment.

Other functional units can be developed to provide higher performance capabilities as may be required in certain applications. The processor is designed to allow new units to be connected to the machine without disturbing the physical hardware, and without impact on the microinstruction format. Floating-point and Fast Fourier Transformer operations are examples of processes which may require specialized units in certain applications. The processor control unit, including the program memory, is shown attached to the crossbar in a manner similar to any other functional unit, and is referenced in the microinstruction for branches and condition sensing in a similar manner.

There are a number of applications in which the performance capabilities of a single SPARC processor will not meet the computational requirements. Typically, in these applications the same algorithm or program is run continuously. The same algorithm or a small number of algorithms are repeatedly used to process the data. In addition for the need for high computational capability, in some applications the input/output requirements can exceed several hundred megabits per second, thus requiring a very flexible and high-performance system I/O structure. Although these processing systems tend to be dedicated in these applications, there is a need to be able to run several types of algorithms on the same processor array. Therefore, there is the need for a reconfigurable structure, and in general, specialized configurations are to be avoided.

An interprocessor communication mechanism is needed which provides the necessary bandwidth between processors. There are several candidate architectures, including fully interconnected systems, shared memory systems, and bus-oriented structures. Recently at CDC another form of communication between processors has been studied, called the Ring System. The Ring Interconnect

System is most often used in low data rate applications such as telephone networks, minicomputer interconnection systems, and in peripheral I/O systems. CDC has been experimenting in using the ring to tightly interconnect high-performance processors. Several examples of candidate Ring System configurations are shown in Figure 4. With the Ring System architecture, the data is passed from one processor to another, and circulates around the ring until being removed by the destination processor. In the CDC design, the data does not pass directly to the internal processor data/instruction network, but instead through a piece of the hardware called the Ring Port, which makes decisions regarding the passage of data. The ring shifts simultaneously in a synchronous manner so that multiple data types can exist on the ring simultaneously. Effectively then, the data bandwidth is multiplied by the number of processors on the ring, providing that the algorithm or computational work can be appropriately structured. In signal processing applications which CDC has investigated, the algorithm can generally be partitioned so that the main data flow takes place between adjacent processors on the Ring. The Ring System represents a relatively low cost, high-bandwidth mechanism for passing data between processors. The form of the Ring System which CDC has been investigating most closely is shown in the lower left-hand corner of Figure 4. This form has two counter-rotating rings in which all processors are connected, and in which data flows in opposite directions. This system can provide up to 1.4 megabits per second of data and control flow in each direction.

An initial approach to interfacing SPARC through PDP-11 equipment at CMU is shown in Figure 5. With this design, the PDP-11 is provided with a Ring Port which has nearly the same capability as the Ring Port contained within the SPARC processor. Through this mechanism, the PDP-11 can communicate with SPARC and also with any of multiple processors which may be included in a Ring System Array.

CABINETRY

An investigation into the chassis, cabinetry, power supply, and cooling apparatus required to house the SPARC processor revealed the existence of a current cabinetry design which is in pilot production for another Control Data

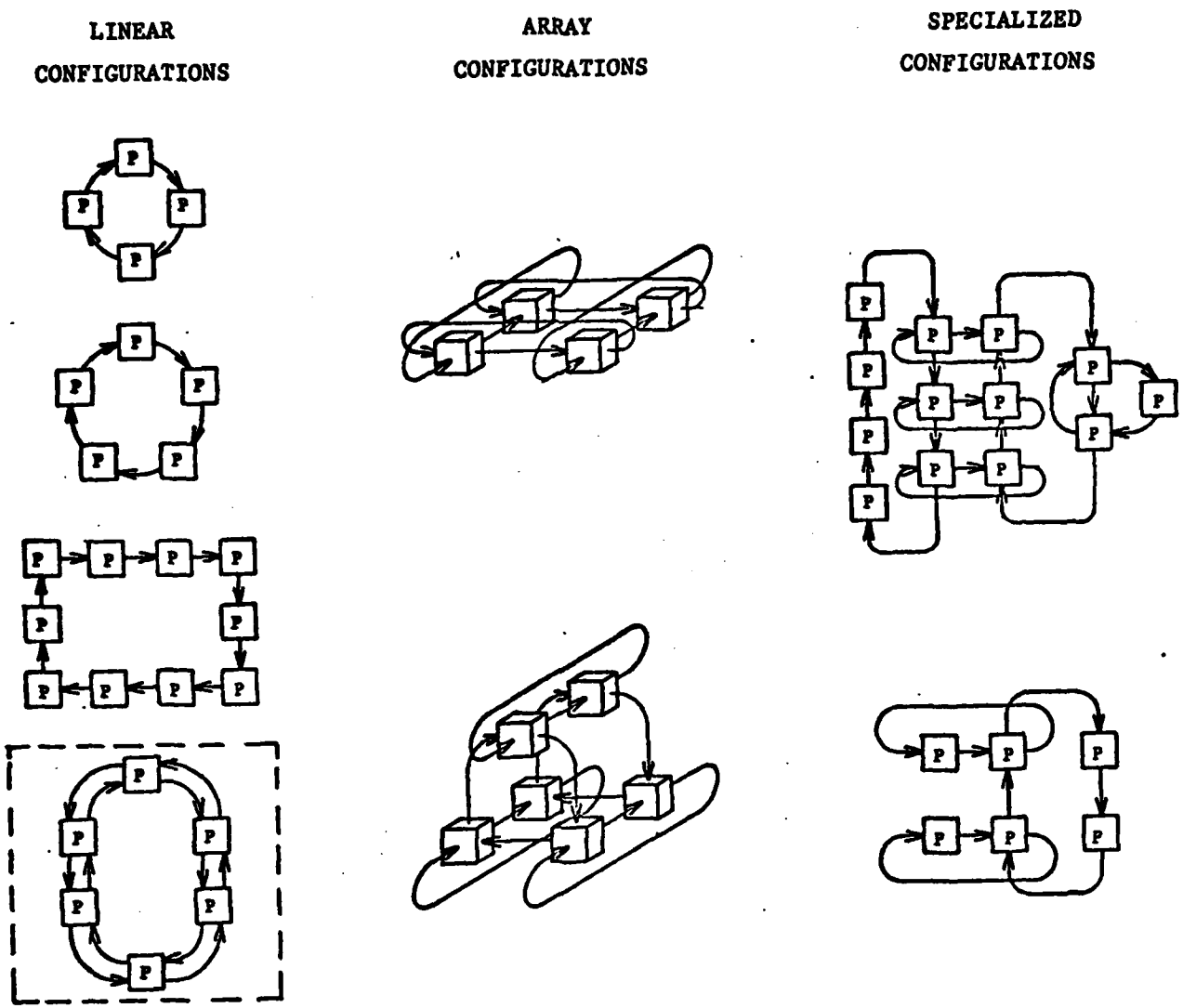


Figure 4. Ring System Configurations

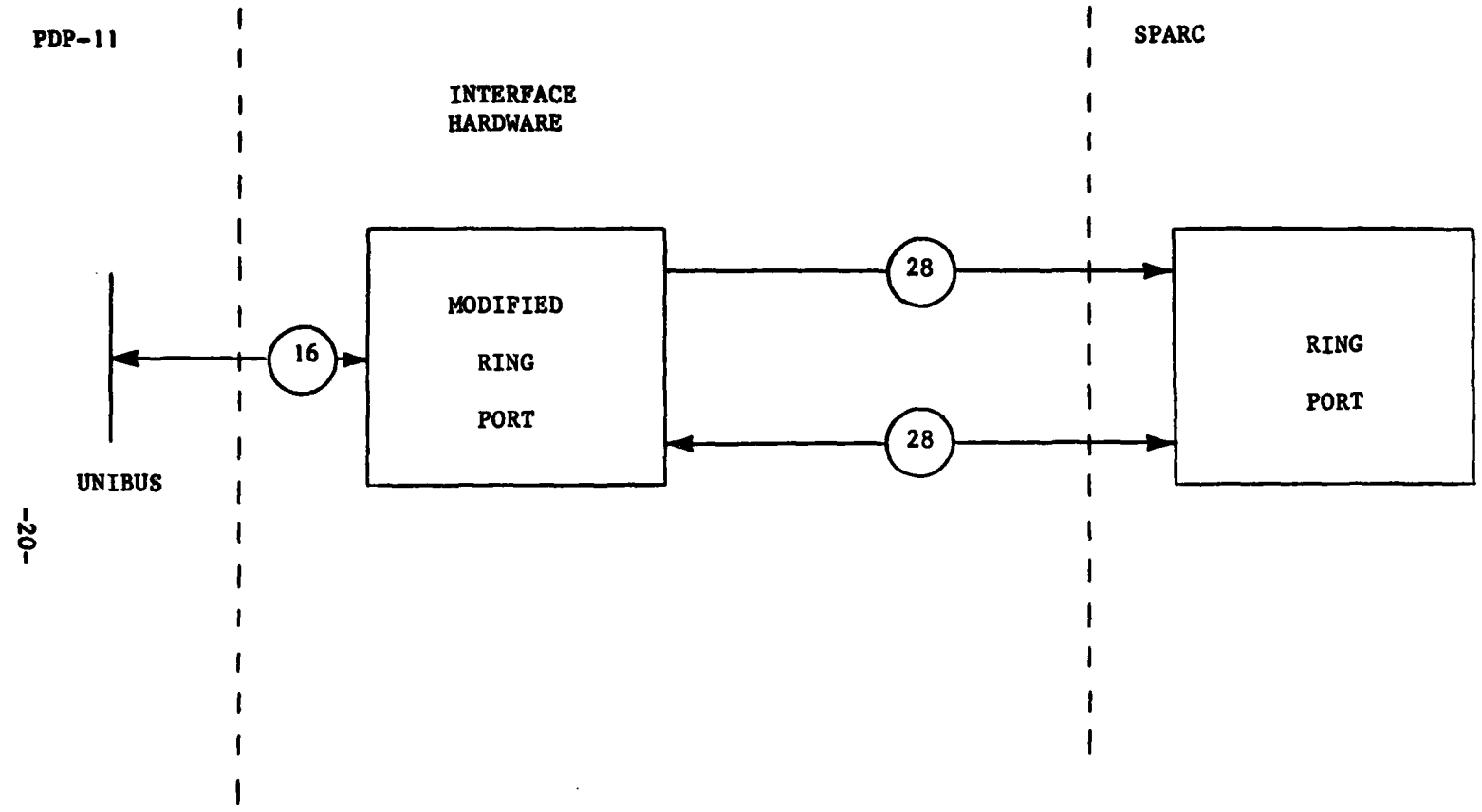


Figure 5. SPARC/PDP-11 Interface

project. This design, which is self-contained, requiring only adequate supplies of AC power and chilled water, was found to be suitable to house the processor as well as provide adequate support for future expansion. The decision was made to utilize this chassis for the initial SPARC machine, thus minimizing the amount of mechanical design on the project. Orders have been placed for piece-parts required to build the cabinet and for fabrication. Delivery of the cabinetry is expected in March of 1979.

SOFTWARE EFFORT

Both CMU and CDC are engaged in developing software to support the SPARC processor. In the case of CMU, the microcode cross-assembly and register-level simulator are designed to operate on the PDP-11 host computer for SPARC under the Unix operating system. At CDC, the microcode cross assembler and register-level simulator will be written in fortran to operate on large CDC GP computers such as 6000, 7000, or CYBER lines. CMU will also develop a library of image understanding algorithms coded for SPARC that will analyze the performance of the hardware on these problems. CDC software also includes diagnostics and development of the basic operating system. CDC software effort is being supported by Control Data and not by the Image Understanding Program. Although the two microcode cross-assemblers will be implemented in different codes, the basic user interaction features are expected to be equivalent. The basic design objectives for the assembler are listed in Table 4. Work on providing a more usable instruction format for the program is continuing. The direction the format development is taking is taking is indicated in Table 5, with examples of coding shown in Table 6. In addition to the low level coding language, considerable work needs to be done on the higher level languages. A language capability is needed to support initial algorithm development. As the algorithm matures, portions of it (kernels) can be converted to higher performance microcode. In addition to these languages, it appears that in many applications a fortran programming capability will be needed.

Design goals for the simulator have been defined and include such features as the ability to operate in both interactive and batch environments, user selected machine configurations, breakpoint features, the ability to run user selected checkpoints and to restart from checkpoint files and various trace and dump options. The main computer implementation language for the CDC version of the simulator shall be fortran and the simulator shall be so coded as to operate on Cyber systems.

As an aid to visualization of the parallel data flows found in an efficient SPARC program, the SPARC data flow graph has been devised. This flow graph is shown in Figure 6. Each symbol on the flow graph represents the SPARC functional unit. Each successive column moving from left to right, represents one instruction step. Lines are drawn from functional unit outputs in one column to functional unit inputs in the next, to represent the crossbar requirements in the receiving column. Such tools allow the programmer to easily visualize the parallel operations which take place in a SPARC processor and to intermesh various related or unrelated operations to achieve maximum functional unit usage.

STATUS

The objective of the current phase of this project is to produce a processor and have it installed at CMU early in the Fall of 1979. At the conclusion of the first six months of the project, the hardware design is approximately 75% complete. The partitioning of SPARC functional units, including the adders, shift/boolean unit, data memories, and multiplier into ECL LSI and MSI arrays has been completed and logic diagram for these units have been prepared. Work is still in progress on design of the control and input/output sections. The gate-level simulation of these units is also approximately 75% complete. The majority of the SPARC electronics hardware consists of existing types of ECL LSI arrays, which have been developed for future and general purpose CDC machines. However, in addition to utilizing fourteen existing arrays types, three new array types are being developed for SPARC. This development is being undertaken to improve the machine design and in particular, to greatly reduce

TABLE 4. MICROCODE ASSEMBLER

- FREE FORMAT INPUT
- PRODUCE ABSOLUTE/RELOCATABLE BINARY
- CONDITIONAL ASSEMBLY
- PROGRAM STATISTICS
- SYMBOL & FUNCTIONAL UNIT CROSS-REFERENCE
- BATCH OR INTERACTIVE
- LOGICAL DIAGNOSTICS
- DATA FILE INITIALIZATION

TABLE 5. GENERAL SOURCE FORMAT

FIELD TYPE	GENERAL FORM
LABEL	1 to 8 characters, beginning with a letter, eg. Part 2
CONSTANT	K = CONSTANT
MAP	DEST = OP (RA, RB, RC, RD/C1, C2, C3, C4)
JUMP	JA (CLK) if (OPR1, R, OPR2)
COMMENT	"COMMENT"

TABLE 6. MICROINSTRUCTION EXAMPLES

```
TOP     K = $3B27     AO = ADD(D2,D3)          "ADD SUMS

                      FO = G4X(,B1)            "WRITE G, LOCATION 4

                      BO = *(,AO)              "CLOCK AO TO BOOLEAN

                      B1 = PASF(FO)            "SHIFT FILE F

                      FO = F5XGX7(BO,B1)       "MOVE TEMP VARIABLES

                      AO = *(LO)               "ADD CROSS PRODUCTS

                      A1 = *(HO)               "   DITTO



        K = SUB2      JK(PUSH)                 "JUMP TO SUBROUTINE SUB2
```

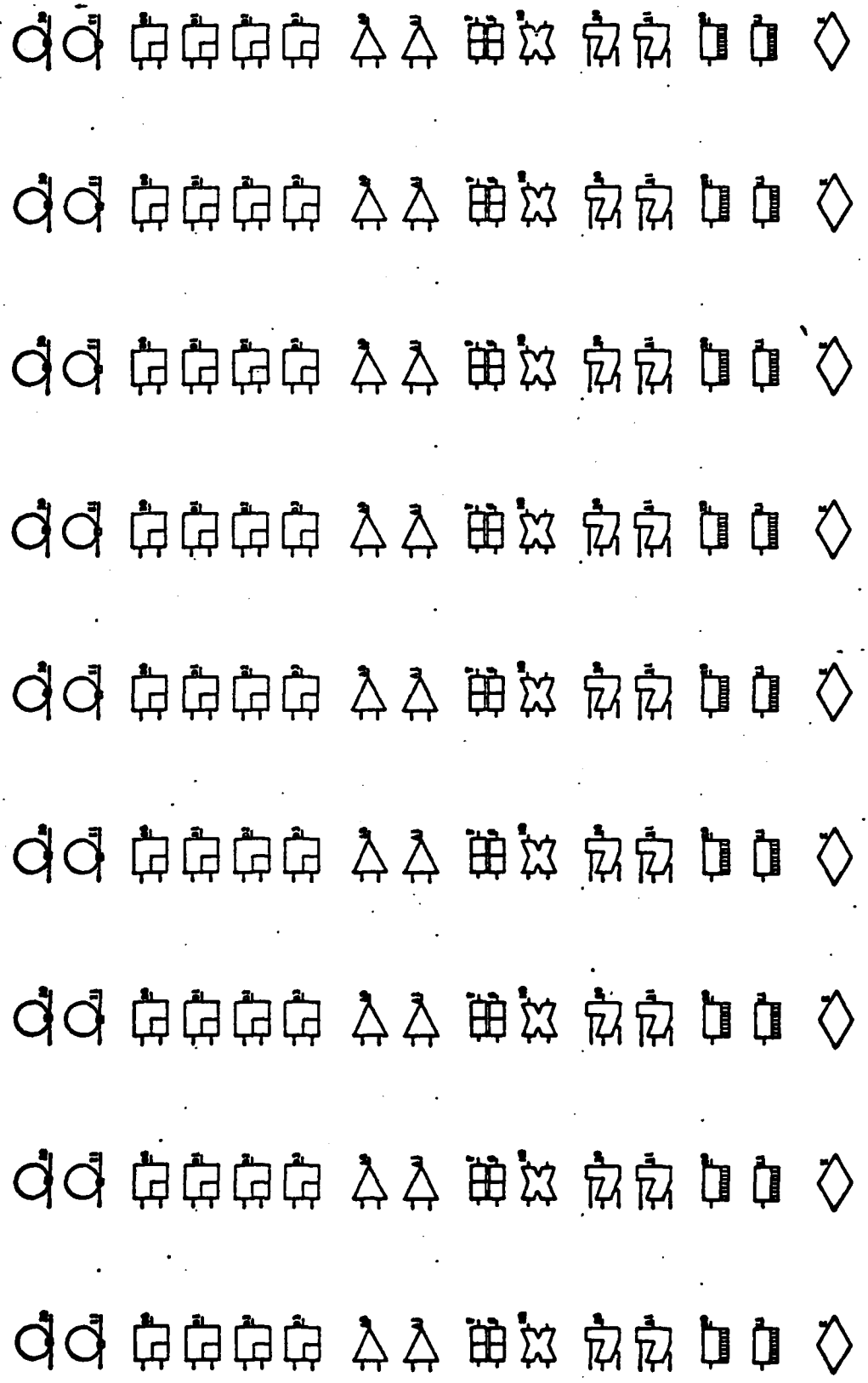


Figure 6. SPARC Flow Graph

the chip count and improve performance. An example of a place in which a new chip type is warranted is in the central data switching data mechanism of the processor. This switch cannot be implemented with existing circuitry without a major sacrifice of machine capability. Since a new array type was necessary, a gate-level of the array has been done. Additional arrays which provide improved capability in the control section are also in process. At present, more than one-half of the machine's electronic components have been ordered. The new array types mentioned above have been placed on order with the CDC Array Development Center. Also on order is the processor cabinet which contains power supply wiring, power supplies, freon cooling equipment, and considerable power distribution and protection mechanisms. This cabinet, developed by CDC for a new product line, has the capacity to hold three SPARC processors and thus provides considerable expansion capabilities for future upgrades in hardware capability at CMU.

Documentation on the SPARC processor has been produced in the form of a processor design specification which is revised, as required, to reflect the latest status of the processor design. In addition, requirements definition documents and preliminary reference manuals are being prepared for the micro-code cross-assembler and the register-level simulator. Coding is now in progress on these two software elements. A great deal of the initial design of the processor diagnostic programs and basic operating systems have also been performed.

CONCLUSION

The cooperative research project between CDC and CMU has developed a new problem-oriented, high-speed digital process architecute for image processing. Under the current project, a processor is being developed which will be capable of approximately 0.2 billion instruction per second execution. In addition to the initial capability of the processor to be interfaced directly to a host PDP-11 machine at CMU, the processor will have the capability of being configured in groups or arrays to provide more processing power if required for more complex algorithms. Such a complete processor system would not only include processors and the host machine, but a hierarchy of memory, including several levels of bulk memory and interfaces to other forms of peripheral equipment.